Event Matching Technique for Multidimensional Range Predicate

Gihong Kim
Dept. of Computer Science & Engineering, Pusan National University, Jangjeon-dong, Geumjeong-gu
Busan, Republic of Korea
buglist@pusan.ac.kr

Bonghee Hong
Dept. of Computer Science & Engineering, Pusan National University, Jangjeon-dong, Geumjeong-gu
Busan, Republic of Korea
bhhong@pusan.ac.kr

Abstract—Event matching technique against lots of multidimensional range predicates is important in the many applications. Existing techniques are restricted to the domain that all the predicates are specified by the user. However, generally some predicates can be omitted by the application requirement. If some predicates are omitted, it is impossible to insert predicates to existing indexing scheme. One of alternatives can be naïve sequential matching that all the events are compared with all the predicates one by one and more advanced alternative is intersection of multi 1-d index. To solve this problem, we propose a novel index scheme for multidimensional range predicates that can be omitted by the application requirement.

Keywords - query index; stream event matching; range predicate index;

I. INTRODUCTION

In recent years, a new kind of emerging application with new data model is widely applied in every field. Such applications generate data in form of data stream rather than persistent data set. For example, sensor applications, network measurements, RTLS (Real-Time Locating System) application and financial ticker etc. these applications process real-time, unbounded, continuous sequence data item, called DSMS(Data Stream Management System).

DSMS brings many research issues and new challenges for data management research society. Lukasz. G and M.T Ozsu etc al surveys the data stream processing issues include continuous query language, data model, algorithmic issues and review recent projects in[12][13]. Among these issues, event matching is way to detect the some specific user interested event which happened in data stream. Event matching condition is presented as conjunctive of predicates. Only when all predicates is matched, we can say the event is happened.

Many research efforts have achieved on event matching processing. The naïve approach on event matching is sequential matching. All the events are compared with all predicates one by one. More advanced alternative approach is creation query index scheme using predicates. To efficient execute event matching, some query index approaches are proposed. The CQI and VCR are query index scheme for moving object data stream. The event matching predicate range is divided into various size rectangles. Such rectangles are inserted in query index scheme. Query index scheme can improve performance of event matching processing. However, existing query index scheme is only suitable for fixed number dimensional predicates. i.e., if some dimensional predicates are omitted, it is impossible to insert predicates to existing index scheme.

In this paper, we consider the problem of event matching for multidimensional range predicate. Firstly, we analyze the event matching range predicates. There are two kinds of event matching condition, the Full Dimensional Query (FDQ) and Sub Dimensional Query (SDQ). Then, we mainly examine SDQ using existing approaches. Compare and analyses the single N-dimensional, Multi 1-dimensional and combination index scheme. The main problem is that unspecified dimensional predicates cannot be presented and inserted by existing approaches. That will cause high memory cost and high insertion cost. To solve these problems, we propose a novel index scheme for multidimensional range predicates that can be omitted part of dimensional rang predicates by the application requirement.

This paper is organized as follows; Section 2 introduces some related work on existing query index structure. Section 3 we clearly define the target environments and explain existing query index problems which we solved in this study. Then, the naïve approaches for query index construction and Infinite Construct Rectangle (ICR) are described in section 4. Finally, we conclude our study at section 5.

II. RELATED WORKS

The continuous queries in DSMS are usually expressed as predicates on a set of attributes. DSMS monitors these predicates against a stream of events which are occurred in the monitored environments. Once an event is matched with all the predicates of a continuous query, proper actions can be taken.

One of the most critical requirements of monitoring continuous queries is the fast matching of events against the predicates. There are several prior works for fast event matching [1, 2, 3, 4] which mostly focus on building predicate indexes with equality-only clauses. However, many predicates contain non-equality range clauses, such as intervals or regions.
For example, continuous queries in our RTLS ALE are usually expressed as cubes in 3D space for monitoring the current location of specific moving objects. In this case, it must retrieve the result from three query indexes, and merge these results as the index is created for each dimension.

Continuous queries indexing techniques for multiple moving objects have been proposed in the literature [5, 6, 7]. In [5], an R-tree based query indexing method for continuous range queries over moving objects was proposed. A safe region for each moving object was defined, allowing an object not to report its location as long as it has not moved outside its safe region. However, determining a safe region requires intensive computation and R-tree is not effective as it is a disk-based approach.

CQI (Cell-based Query Index) [6] was a main memory-based indexing approach and was shown to perform better than the R-tree based query index. In [6], the monitoring area is partitioned into cells. Each cell maintains two query lists: full and partial. The full list stores the IDs of the queries that completely cover the cell, while the partial list keeps those that partially intersect with the cell. During query reevaluation, these lists are used to find all the queries that contain an object position. However, this approach has some drawbacks. The object locations must be compared with the range query boundaries in order to identify those queries that truly contain an object. Because of that, it cannot allow query reevaluation to take advantage of the incremental changes in object locations. On the other hand, a range query can be decomposed into a large number of grid cells, incurring a large storage overhead. Moreover, grid cells of a fixed size L × L, where L > 1 cannot perfectly cover every range query.

VCR (Virtual Construct Rectangle) index [7] was also a main memory-based approach and presented for incremental processing of continuous range queries over moving objects. It was shown to outperform the CQI approach. In [7], it uses one or more virtual constructs to decompose the query regions. A set of VCRs is predefined, each with a unique ID. The query ID is then inserted into the ID lists associated with those covering VCRs. The use of VCRs provides an indirect and cost-effective way of pre-computing the search result for any give event and the event matching or searching is very efficient. However, many of the VCRs defined are redundant and unnecessarily degrading the index search and query reevaluation times and the number of VCR covering a given data point can be large which also leads to increase in search time.

It is difficult to construct an effective index for multidimensional range queries. There are other indexing techniques of moving objects have been proposed [8, 9, 10]. Different constraints are usually imposed to reduce the overhead caused by location updates and queries insertion. The more complex problem of continuous query indexing is that the queries contain some predicates which may represent an infinite query range on data dimensions [11].

III. TARGET ENVIRONMENT AND PROBLEM DEFINITION

A. Target Environment

Whenever we use the real time system like some tag such as RFID, RTLS, sensor or some kind of network traffic, stock flow or some communication system like telecommunication and web access the system need to process the streaming data from different sources. The source may be traffic flow data, spacio-tempo, RFID, sensor or RTLS data or other sources which provide continuous and huge data. The data stream management systems are used to process the streaming data to provide the real time application. And for this type of application, continuous query is very important which mainly works on the events generated by the streaming data source. To process the real time data for continuous query data stream management system (DSMS) rules have to be followed or special type of query processing for streaming data have to be maintained such as window processing or query range processing as all the real time data are not available at the processing time.

One such type of query processing is the continuous query index based on dimension. The dimension of continuous query depends on the application criteria and requirements. Based on the application requirements continuous queries can be divided as 1-D query, 2-D query and 3-D query or fully dimensional query. For example, one of the dimensional continuous queries is the spatio-temporal DSMS query. In this type of query processing, the query’s predicate are the value of X-axis and Y-axis and the tag ID as tag ID specify a specific tag and the value of X-axis and Y-axis defines the position of the tag. So in this type of spacio-temporal continuous query, if all the predicates (X-axis, Y-axis and tag ID) are specified then this type of query is called the 3-D query, if two of them are specified then it’s called the 2-D query and if only one of them specified then it’s called the 1-D query. The following figure shows these three types of queries.

![Figure 1. Dimensions of continuous query](image-url)
Another example of dimensional continuous query is the semi-passive sensor tag data management. Sensor tag is used to monitor the environment of the product where it is kept. In this type of system, tag ID, reader ID and the sensing value are the important parameters to process. So in this type of streaming data processing, if all the predicate (tag ID, reader ID, sensing value) are specified in the query, then this type of query is known as 3-D, if two of them specified then this type is called the 2-D query and if only one of them present then this type is known as 1-D query.

In streaming data management system, if any continuous query specified all the predicates then we called this type of query as Full Dimensional Query (FDQ) and if not all the predicate specified then we called this type of query as Sub Dimensional Query (SDQ). So from the above example, both 1-D query and 2-D query fall to the category of SDQ and 3-D query falls to the category of FDQ.

To manage the streaming data of the real-time application system, the Data Stream Management System has to process both FDQ and SDQ.

B. Problem Definition

The important characteristics of event streaming data are that the event data arrive in the system continuously, rapidly and also unboundedly. Traditional data index that are used to improve the processing of streaming data suffer a lot from frequent updates. And the query to the streaming system is not the one time query rather it process streaming continuous query over the streaming data that run over a period of time. In this streaming data processing environment continuous query index are used to process the user continuous query faster avoid updating of the index. This approach build an index for the continuous queries which is suitable for streaming data processing as these remain active for a period of time. The existing techniques of processing the continuous queries are restricted to the fact that all the predicate of the query be defined by the user. But different application modified the queries based on its requirements and modified queries sometimes does not have all the predicate that defined by the user for processing the continuous queries. But the approach allows only queries which have the all predicate defined by the user to store in the continuous query index. This is because, if sub-dimensional queries are stored in the CQI index, then one or more of its fields need unlimited space to store the unlimited data index. And the processing the query index with unlimited value, it suffers a lot. That is, this suffers from both memory and processing time problem. So if some of the predicate of the queries is omitted by the system for the application requirement then the approach does not support this type of queries to store in the query index. Thus the sub query is impossible to process by the existing query indexing technique.

IV. ICR INDEX FOR 3D RANGE PREDICATES

In this section, the contents are two folds. First, we make an analysis of the data called Finite data (FD) and iNfinite Data (ND) for building continuous query index. Second, we analyze the simple approach like Single Multi-D Index, Multi 1-D Index, and Index by combining all the attributes.

A. Analysis of Data

In case of constructing continuous query index, each user defined query is the data which used to build the index. As we mentioned before, queries can be classified into two categories which are called Full Dimensional Query (FDQ) and Sub Dimensional Query (SDQ) respectively. We define FDQ as Finite Data (FD) and SDQ as the iNfinite Data (ND) in this work for building the index. The detailed analysis of FD and ND are as follows.

![Finite Data (FD) for Q1](image)

We define the full dimensional query as the Finite Data (FD) in building the query index, as a result of that all the attributes in this type of query are under certain limit, or the value of each attribute in this type of query is inside certain range. Considering that user defined a FDQ called Q1 as “report event while tid 10 is in the region of x: 10~20, y: 20~30 (x, y, here are the coordinates for location)”. As specified by Q1, x attribute is limited to 10 and 20, y attribute is limited to 20 and 30, and tid attribute equals to 10. After inserting the Q1 into a three dimension index, it is shown as a rectangle (as Fig.2 shown) in the three dimensional query index space. The area is a finite value. Even if the tid value is inside certain range, then Q1 is a rectangular block in the three dimensional space, its volume is still a finite value. Therefore, we can conclude that full dimension query always has a finite volume in its full dimension index space. And the Full Dimensional Query (FDQ) here is defined as the Finite Data (FD) for building the continuous query index.

We define the Sub Dimensional Query (SDQ) as the iNfinite data in building the query index. While the FDQ has limitation on all the attributes, the SDQ only has limitation in some of the attributes. In other words, some attributes is infinite in the full dimensional index space. One case is that user defined SDQ called Q2 as “report event when any tid is in the region of x: 10~20, y: 20~30”. Attribute “tid” has no limitation at all, showing by Fig. 3, the volume of the rectangular block goes to infinity. Another case of the iNfinite Data (ND) is that user defined a SDQ called Q3 as “report event when tid 10 is in any region” is defined by the user. Attribute “x” and “y” have no limitation in its value range, though the tid is limited to 10. However, the area of the Q3 in the three dimensional space goes to infinity. Therefore, we define the Sub Dimensional Query (SDQ) as iNfinite Data (ND) for building the continuous query index.
Through analyzing the data for constructing the continuous query index, we define the FDQ as the Finite Data (FD) for building the index while the SDQ is the iNfinite Data (ND).

B. Naïve Approach and Infinite Construct Rectangle (ICR)

Here, we make an analysis of three naïve approaches which are 1) Single multi-D index, 2) Multi 1-D index, and 3) Combination Index. In the following analysis we take the three dimensions (3-D) as an example for easy understanding. However, the problem that exists in the 3-D index also occurs in the multiple dimensional index whose dimension number is larger than 3. Our analysis is not limited on three dimensional index building. As Fig. 4 shows three naïve approaches in a simple way. Considering three attributes “a”, “b”, and “c” are used to build index. Triangle is representing for the index that we are going to build. As Fig. 4 shows, 1) Single multi-D index only has one index with three dimension on attributes “a”, “b”, and “c”, 2) Multi 1-D index has three one dimension indexes, each index on one attribute, and 3) Combination index scheme has 7 indexes, three one dimension indexes, three two dimensional indexes, and one three dimension index.

As the left up corner of Fig. 4 showing, first, the single multi-D index, this approach builds an index for all attributes. In this example, it likes building a three dimensional space. There are two main problems for this approach to insert iNfinite Data (ND) query data into the continuous query index (using the Cell-based Query Index). One is the time problem, this approach needs to insert \(4,294,967,296\) times for one ND. The other one is the memory problem, the memory cost will be \(4,294,967,296 \times 32\) (cell node info size) = \(137,438,953,472 = 16\)GB. Second, as the right up corner of Fig. 4 showing, the Multi 1-d index, it builds several indexes, each index correspond to an attribute. The problem of this approach is that the maintenance and intersection cost is too high. As there are \(n\) indexes under maintaining, and intersection is the one type of most expensive operations between the indexes, and user queries about data with more than one attribute is inevitable. Third, see the bottom part of the Table 1, the combination index this approach builds several indexes also. \(2^n-1\) indexes will be needed in this approach. If \(3-d\) then \(2^3-1 = 7\) indexes are needed. The problem of this approach is the same as the multi-D index, it is impossible to insert SDQ which may also be called as ND to query index.

<table>
<thead>
<tr>
<th>Type</th>
<th>Insertion cost</th>
<th>Search cost</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single n-D</td>
<td>Good</td>
<td>Poor</td>
<td>Impossible</td>
</tr>
<tr>
<td>Multi 1-D</td>
<td>(n) index</td>
<td>Bad for Intersection cost</td>
<td>Moderate</td>
</tr>
<tr>
<td>Combination</td>
<td>(\infty)</td>
<td>Good</td>
<td>Impossible</td>
</tr>
</tbody>
</table>

As a conclusion (shown in Table 1) to the naïve approaches: 1) Single n-D Index has the infinite insertion cost thought the search cost is good, therefore this approach is impossible to solve the proposed problem. 2) Multi 1-D index, the index number is equals to the attributes number, if the attributes number is large and the insertion cost will be very high, and due to the intersection cost the search cost is also high. Therefore this approach is moderate but not good. 3) Combination index has the same problem with single n-D index, so that it is also impossible to solve the problem.

Here, we show Infinite Construct Rectangle (ICR) to solve multidimensional query problem in Fig 5.
V. CONCLUSION

In this paper, we propose a event matching technique for multidimensional range predicate. The continuous queries in DSMS are usually expressed as predicates on a set of attributes. DSMS monitors these predicates against a stream of events which are occurred in the monitored environments. Existing techniques are restricted to the domain that all the predicates are specified by the user. However, generally some predicates can be omitted by the application requirement. And we analyze the event matching range predicates. There are two kinds of event matching condition, the Full Dimensional Query (FDQ) and Sub Dimensional Query (SDQ). One of the most critical requirements of monitoring continuous queries is the fast matching of events against the predicates. VCR index was also a main memory-based approach and presented for incremental processing of continuous range queries over moving objects. But the approach allows only queries which have the all predicate defined by the user to store in the continuous query index. This suffers from both memory and processing time problem. So if some of the predicate of the queries is omitted by the system for the application requirement then the approach does not support this type of queries to store in the query index.

ACKNOWLEDGMENT

This work was supported by the Industrial Strategic technology development program (65300604, On-site Applicable Intelligent Software and Common Standard Platform Development) funded by the Ministry of Knowledge Economy(MKE, Korea)

REFERENCES


